Operační systémy 1

7. Cvičení

Radek Janoštík

KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Úkoly z minula – jak to šlo?

- Kolik se vám toho podařilo udělat?

```
1 void print_row(int n, char c)
2 void print_rect(int rows, int cols)
3 unsigned int factorial(unsigned int n)
4 char *my_strdup(char *s)
5 unsigned int fib(unsigned short n)
6 void print_facts(unsigned char n)
```

- Dalo něco zabrat?
- Nějaké dotazy?
- Prezenčka (až přijdou všichni :))

Informace k písemce

- Termín: Příští cvičení – 30. 3. 2026
- Obsah: Všechna probraná cvičení (i to dnešní)
  - Projděte si všechny texty
  - Dodělejte si všechny úkoly k procvičení
  - Snažte se věci pochopit
- Složení testu:
  - Několik otázek na „pochopení učiva“
  - Programování „na papír“ – rozsah několik řádků kódu
- Jakékoliv učební materiály či používání elektronických zařízení nebude povoleno

Systémová volání

- Zatím jsme využívali luxusu propojení assembleru a jazyka C
  - Např. pro výstup jsme používali funkci `printf`
  - Pro spouštění programu a volání funkci jsme využili `main` a jazyk C
- Bez jazyka C musíme přímo interagovat s operačním systémem
- OS poskytuje rozhraní (systémová volání) pro základní operace
  - Čtení a zápis do souborů
  - Ukončení programu a předání návratové hodnoty
  - Práce se souborovým systémem
  - ...

Realizace systémového volání v Linuxu (AMD64)

- Využití systémového volání je velmi podobné jako volání funkcí (ale jiná konvence)

1. Každá služba OS (např. otevření souboru, změna adresáře) je identifikována číslem, které je uloženo registru `rax`.
2. Argumenty předávané jádru (např. název souboru, příznaky) jsou uloženy v registrech `rdi, rsi, rdx, r10, r8, r9` (v tomto pořadí).
3. Služba OS je zavolána instrukcí `syscall`.
4. Návratová hodnota je uložena v registru `rax` (záporné hodnoty indikují chybu), obsah registrů `rcx` a `r11` může být změněn, obsah dalších registrů je zachován.

- Seznam systémových volání nalezneme v dokumentaci, přehledně např. na `https://www.chromium.org/chromium-os/developer-library/reference/linux-constants/syscalls/`

Minimální program (bez jazyka C)

- U minimálního programu musíme vyřešit pouze dvě věci:
  - Který kód se má začít vykonávat.
  - Jak korektně ukončit program.

```
1  global _start
2
3  SYS_EXIT      equ 60
4
5  section .text
6
7   _start:
8        mov rax, SYS_EXIT ; zvolíme službu OS - 60 = sys_exit
9        mov rdi, 42       ; nastavime navratovou hodnotu
10       syscall           ; zavoláme službu OS
```

Překlad a spuštění

- Objektový soubor sestavíme stejně jako dříve (`nasm`)
- Pro spustitelný soubor nevyužijeme `gcc`, použijeme přímo linker `ld`

```
tutorial07: tutorial07.o
     ld -o tutorial07 tutorial07.o
tutorial07.o: tutorial07.asm
     nasm -f elf64 tutorial07.asm
```

- Program nedělá nic. Pro ověření návratové hodnoty využijeme `shell`

```
$ ./tutorial07
$ echo $?
42
```

Hello World bez jazyka C

```
1  global _start
2  ; deklarace konstant
3  SYS_WRITE      equ 1  ; systemove volani pro zapis do souboru
4  SYS_EXIT       equ 60 ; systemove volani pro ukonceni programu
5  STDOUT         equ 1  ; deskriptor souboru standardniho vystupu
6  STR_HELLO_LEN  equ 13 ; delka vypsaneho retezce
7  ; spustitelny kod
8  section .text
9  _start:
10     mov rax, SYS_WRITE ; vypsani retezce Hello World
11     mov rdi, STDOUT
12     mov rsi, str_hello
13     mov rdx, STR_HELLO_LEN
14     syscall
15
16     mov rax, SYS_EXIT ; ukonceni programu
17     mov rdi, 42
18     syscall
19 ; (inicializovana) data programu
20 section .data
21 str_hello:
22     db "Hello World!", 10
```

Hello World – rozbor

- Systémové volání `write` má 3 argumenty
  - Deskriptor souboru, do kterého budeme zapisovat (1)
  - Řetězec, který se má do souboru zapsat (sekce `.data`)
  - Délka tohoto řetězce (13)
- Pro data v kódu vyčleněny 3 části:
  - `.data` – obecná data
  - `.rodata` – data jen pro čtení
  - `.bss` – neinicializovaná data
- Využití sekce `.data` a pseudoinstrukce `db`
  - Po spuštění načteny ze souboru do paměti $\Rightarrow$ instrukce pro práci s pamětí

Čtení dat ze standardního vstupu

- Standardní vstup je v Linuxu opět soubor ⇒ systémové volání pro čtení ze souboru `read`
  - Deskriptor souboru: 0
  - Data načtena do zadané paměti → sekce `.bss` (neinicializovaná data)
  - Při chybě vrací zápornou hodnotu
  - Při úspěchu vrací počet bajtů, které se podařilo úspěšně přečíst
- Ladění programu je komplikované. Můžeme si pomoci nástrojem `strace`
  - Ukazuje, jaká systémová volání se (kdy) volala
  - Ukazuje i vstupní a návratové hodnoty

```
echo "abc" | strace ./tutorial07
execve("./tutorial07", ["./tutorial07"], 0x7ffc3a341dc0 /* 100 vars */) = 0
read(0, "abc\n", 64)
 = 4
write(1, "abc\n", 4abc
)
 = 4
exit(0)
 = ?
+++ exited with 0 +++
```

Úkol (1/2)

1. Vytvořte program `rect`, který na terminál vypíše obdélník složený ze znaků $\star$ o stranách 20×5.

2. Vytvořte program `mypwd`, který se bude chovat podobně jako standardní unixový příkaz `pwd` a vypíše na standardní výstup plnou cestu k aktuálnímu adresáři. Jaký je aktuální adresář zjistíte pomocí systémového volání `getcwd`.

3. Vytvořte program `mypwd2`, který vypíše jméno aktuálního adresáře, tj. jméno za posledním znakem `'/'`.

Úkol (2/2)

4. Upravte poslední ukázkový příklad tak, aby vracel počet řádků přečtených ze standardního vstupu. Tyto úpravy provádějte postupně.
   - Spočítejte řádky na vstupu a výsledek vraťte v návratovém kódu.
   - Spočítejte řádky a jejich počet vypište na standardní výstup.
   - Upravte program, aby pracoval s libovolně velkým vstupem, tj. zpracovával vstup, dokud systémové volání read nevrátí 0 nebo zápornou hodnotu.